Package 'linbin'

October 13, 2022

Version 0.1.3

Date 2021-04-16

Type Package

Depends R (>= 3.0.1)

License AGPL-3

Title Binning and Plotting of Linearly Referenced Data

Description Short for 'linear binning', the linbin package provides functions for manipulating, binning, and plotting linearly referenced data. Although developed for data collected on river networks, it can be used with any interval or point data referenced to a 1-dimensional coordinate system. Flexible bin generation and batch processing makes it easy to compute and visualize variables at multiple scales, useful for identifying patterns within and between variables and investigating the influence of scale of observation on data interpretation.

LazyData true

Imports stats, utils, graphics, grDevices

Suggests knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/ezwelty/linbin

BugReports https://github.com/ezwelty/linbin/issues

RoxygenNote 7.1.1

NeedsCompilation no

Author Ethan Z. Welty [aut, cre],
Christian E. Torgersen [ctb] (author support and guidance),
Samuel J. Brenkman [ctb] (elwha and quinault datasets),
Jeffrey J. Duda [ctb] (elwha dataset),
Jonathan B. Armstrong [ctb] (fishmotion dataset)

Maintainer Ethan Z. Welty <ethan.welty+linbin@gmail.com>

Repository CRAN

Date/Publication 2021-04-20 12:20:06 UTC

R topics documented:

`as_events` *Coerce to an Event Table*

Description

Attempts to coerce an object to an event table.

Usage

```
as_events(x, ...)

## S3 method for class 'numeric'
as_events(x, ...)

## S3 method for class 'POSIXt'
as_events(x, ...)

## S3 method for class 'Date'
```

```
as_events(x, ...)

## S3 method for class 'matrix'
as_events(x, from.col = 1, to.col = NULL, ...)

## S3 method for class 'data.frame'
as_events(x, from.col = 1, to.col = NULL, ...)
```

Arguments

x	Object to be coerced to an event table.
...	Additional arguments passed to or used by methods.
from.col, to.col	Names or indices of the columns in x containing the event endpoints. Values are swapped as needed to ensure that to > or = from on all rows. If NULL, to.col defaults to from.col + 1 (if column exists) or from.col.

Methods (by class)

- numeric: Expands a numeric vector into two columns of event endpoints.
- POSIXt: Coerces to numeric before dispatching.
- Date: Coerces to numeric before dispatching.
- matrix: Converts the matrix to a data frame, then calls the data.frame method.
- data.frame: Renames from.col and to.col to "from" and "to" as needed. Since these column names must be unique, other columns cannot also be called "from" or "to".

See Also

events for creating event tables and read_events for reading files as event tables.

Examples

```
as_events(1)
as_events(1:5)
as_events(cbind(1:5, 1:5), 1, 2)
as_events(data.frame(x = 1, start = 1:5, stop = 1:5), "start", "stop")
```

---

crop_events *Crop Events*

Description

Crops events to the specified intervals. Events are cut at interval endpoints and any whole or partial events lying outside the intervals are removed.

Usage

```
crop_events(e, crops, scaled.cols = NULL)
```

Arguments

e	An event table.
crops	An event table specifying the intervals for cropping. Point intervals are allowed, and will create new point events where they intersect the interior, but not the endpoints, of line events.
scaled.cols	Names or indices of the columns of the event table to be rescaled after cutting (see cut_events). Names are interpreted as regular expressions (regex) matching full column names.

See Also

cut_events for only cutting events.

Examples

```
e <- events(c(0, 10, 20), c(10, 20, 30), x = 10)
crop_events(e, events(c(0, 15)))
crop_events(e, events(c(0, 5, 15)))
crop_events(e, events(c(0, 5, 15)), scaled.cols = "x")
crop_events(e, events(c(0, 5, 5, 15)), scaled.cols = "x")  # creates new points inside lines
crop_events(e, events(c(0, 10, 10, 15)), scaled.cols = "x") # but not at line event endpoints
```

---

cut_events *Cut Events*

Description

Cuts events at the specified locations.

Usage

```
cut_events(e, cuts, scaled.cols = NULL)
```

Arguments

e	an event table.
cuts	the cut locations. Can be either a numeric vector or an event table. If an event table that contains points, point events will be created where they intersect the interior, but not the endpoints, of line events in e.
scaled.cols	names or indices of the event table columns to be scaled to their new length after cutting. Names are interpreted as regular expressions (regex) matching full column names.

Details

Line events straddling cut locations are cut into multiple event segments. Columns `scaled.cols` are scaled by the fraction of the original event length in each resulting event (which assumes that these variables were uniformly distributed over the original interval). To have a record of the parents of the resulting event segments, append an unique identification field to the event table before calling this function.

See Also

`crop_events` for both cutting and removing events.

Examples

```
e <- events(c(0, 10, 20), c(10, 20, 30), x = 10)
cut_events(e, events(c(0, 5, 15)))
cut_events(e, events(c(0, 5, 15)), scaled.cols = "x")
cut_events(e, events(c(0, 5, 5, 15)), scaled.cols = "x")  # creates new points inside lines
cut_events(e, events(c(0, 10, 10, 15)), scaled.cols = "x") # but not at line event endpoints
```

---

elwha *Elwha River Survey*

Description

An event table containing the results of a survey of the Elwha River (Washington, USA) carried out in August-September 2008. Both physical variables and fish counts were collected.

Format

A data frame with 249 rows and 33 variables.

Details

- from, to - distance upstream from the river mouth [km]
- unit.length - unit length [m]
- unit.type - unit type (P = pool, GP = glide-like pool, G = glide, GR = glide-like riffle, R = riffle)
- channel.type - channel type (1 = main, 2 = secondary)
- mean.depth - mean depth [m]
- max.depth - max depth [m]
- mean.width - mean wetted width [m]
- bedrock - bedrock substrate [%]
- boulder - boulder substrate [%]
- cobble - cobble substrate [%]

- gravel - gravel substrate [%]
- sand - sand substrate [%]
- silt - silt substrate [%]
- overhang.cover - channel banks with overhanging vegetation [%]
- boulder.cover - channel area covered by boulders [%]
- jams - number of log jams
- jam.area - total area of log jams [m^2]
- SACO.10/20/30/40/total - number of Bull Trout (*Salvelinus confluentus*) sized 10 - 20 cm / 20 - 30 cm / 30 - 40 cm / > 40 cm / total, respectively.
- ONXX.10/20/30/40/total - number of trout (*Oncorhynchus sp.*) sized 10 - 20 cm / 20 - 30 cm / 30 - 40 cm / > 40 cm / total, respectively.
- SAFO - number of Brook Trout (*Salvelinus fontinalis*)
- ONTS - number of Chinook Salmon (*Oncorhynchus tshawytscha*)
- ONNE - number of Sockeye Salmon (*Oncorhynchus nerka*)
- LATR - number of Pacific Lamprey (*Lampetra tridentata*)
- ONKI - number of Coho Salmon (*Oncorhynchus kisutch*)

Source

Brenkman, S. J., J. J. Duda, C. E. Torgersen, E. Z. Welty, G. R. Pess, R. Peters, and M. L. McHenry. 2012. A riverscape perspective of Pacific salmonids and aquatic habitats prior to large-scale dam removal in the Elwha River, Washington, USA. Fisheries Management and Ecology 19:36-53. DOI: doi: 10.1111/j.13652400.2011.00815.x

---

`events` *Event Tables*

Description

Creates an event table, a custom `data.frame` used throughout the `linbin` package to store and manipulate linearly referenced data. Each row includes an event's endpoints `from` and `to` (which can be equal, to describe a point, or non-equal, to describe a line) and the values of any variables measured on that interval.

Usage

```
events(from = numeric(), to = NULL, ...)
```

Arguments

`from, to`	Event endpoints, in any format coercible to single data frame columns. `from` and `to` are swapped as needed so that `to` > or = `from` on all rows. If `from` is the only non-empty argument, `as_events` is dispatched for object coercion.
`...`	Additional arguments, either of the form `value` or `tag = value`, to be passed directly to `data.frame` following `from` and `to`. Component names are created based on the tag (if present) or the deparsed argument itself.

Details

Event endpoints (and any additional arguments) are coerced to a data frame with `data.frame`, then coerced to an event table with `as_events`. A valid event table has two columns named "from" and "to" containing only finite numeric values (i.e., no `NA`, `NaN`, or `Inf`) and ordered such that `to` > or = `from`. `is_events` tests for these requirements. The other columns in the event table can be of any type supported by the `data.frame` class.

Value

An event table, the `data.frame` object used by `linbin` to describe interval data.

See Also

`data.frame`.

`as_events` and `read_events` for coercing objects and files to event tables, `is_events` to validate event tables.

Examples

```
events(1, 5)
events(1:5)
events(c(0, 15, 25), c(10, 30, 35), x = 1, y = c('a', 'b', 'c'))
```

---

event_coverage *Event Coverage*

Description

Returns the intervals over which the number of events is always one or greater.

Usage

```
event_coverage(e, closed = TRUE)
```

Arguments

`e`	An event table.
`closed`	Logical value indicating whether events should be interpreted as closed intervals. If `TRUE`, coverage is continuous at breaks between two adjacent events.

Details

If `closed = TRUE`, breaks between adjacent events are dropped. If `closed = FALSE`, breaks between adjacent events are retained, including point events on line event endpoints. Duplicate points are dropped in both cases.

See Also

event_gaps for gaps (the inverse of coverage), event_range for range (coverage with gaps ignored).

Examples

```
e <- events(c(1, 2, 4, 8), c(3, 4, 5, 10))
event_coverage(e, closed = TRUE)  # retains breaks
event_coverage(e, closed = FALSE) # drops breaks
e <- events(c(0, 2, 2, 2, 8, 10), c(0, 2, 2, 6, 10, 10))
event_coverage(e, closed = TRUE)  # retains isolated points
event_coverage(e, closed = FALSE) # retains isolated points and points adjacent to lines
```

---

event_gaps *Event Gaps*

Description

Returns the intervals over which there are no events.

Usage

```
event_gaps(e, closed = TRUE, range = NULL)
```

Arguments

e	An event table.
closed	Logical value indicating whether events should be interpreted as closed intervals. If TRUE, no gaps are returned at breaks between two adjacent events.
range	An event table specifying, by its event_range, the interval within which to check for gaps. If NULL, the range of e is used.

See Also

event_coverage for coverage (the inverse of gaps), fill_event_gaps for filling gaps with empty events.

Examples

```
event_gaps(events(c(1, 3, 5), c(2, 4, 5)))    # gaps between events
event_gaps(events(1:5))                       # no gaps
event_gaps(events(1:5), closed = FALSE)       # gaps at breaks
event_gaps(events(1:5), range = events(0, 6)) # gaps to edge of range
```

event_midpoints *Event midpoints*

Description

Event midpoints

Usage

```
event_midpoints(e)
```

Arguments

e	Event table.

Examples

```
e <- events(c(0, 10, 15, 25, 30), c(10, 20, 25, 40, 30))
event_midpoints(e)
```

event_overlaps *Event Overlaps*

Description

Returns the number of events on each interval. Useful for sampling the original data with `sample_events` at the highest possible resolution that nevertheless flattens overlapping events.

Usage

```
event_overlaps(e)
```

Arguments

e	An event table.

Details

Point events are preserved and line events are cut as necessary at the endpoints of other point or line events.

Value

An endpoint-only event table with column "n" listing the number of overlapping events on that interval.

See Also

event_coverage.

Examples

```
e <- events(c(0, 10, 15, 25, 30), c(10, 20, 25, 40, 30))
event_overlaps(e)
```

---

event_range *Event Range*

Description

Returns the minimum and maximum endpoints of all the events in an event table.

Usage

```
event_range(e)
```

Arguments

e	An event table.

See Also

event_coverage for an alternative that accounts for gaps.

Examples

```
event_range(events(1:5))            # no gaps
event_range(events(c(1,5), c(1,5))) # gaps
```

---

fill_event_gaps *Fill Event Gaps*

Description

fill_event_gaps fills gaps below a maximum length with empty events. collapse_event_gaps shifts event endpoints to close gaps below a maximum length.

Usage

```
fill_event_gaps(e, max.length = Inf)

collapse_event_gaps(e, max.length = Inf)
```

Arguments

e	An event table.
max.length	The maximum length of gaps to be filled or closed.

See Also

event_gaps

Examples

```
e <- events(c(1, 4), c(2, 5), x = 1)
fill_event_gaps(e)
fill_event_gaps(e, max.length = 1)
collapse_event_gaps(e)
collapse_event_gaps(e, max.length = 1)
```

---

find_intersecting_events
Find Intersecting Events

Description

Returns a logical matrix indicating whether or not each pair of events intersect.

Usage

```
find_intersecting_events(ex, ey, equal.points = TRUE, closed = FALSE)
```

Arguments

ex, ey	Event tables.
equal.points	If TRUE, equal-valued points are considered intersecting. This is always TRUE if closed = TRUE.
closed	If TRUE, events are interpreted as closed intervals and events sharing only an endpoint are reported as intersecting.

Value

A logical matrix with ey events as rows and ex events as columns.

Examples

```
ex <- events(c(0, 5, 5, 10))
find_intersecting_events(ex, events(5), equal.points = FALSE) # equal points don't intersect
find_intersecting_events(ex, events(5), equal.points = TRUE)  # equal points do intersect
find_intersecting_events(ex, events(5), closed = TRUE)        # adjacent events intersect
find_intersecting_events(ex, ex)
```

fishmotion *Fish Movements*

Description

A pair of event tables (in a list) documenting the movements of tagged Coho Salmon (*Oncorhynchus kisutch*) in Bear Creek (Southwest Alaska, USA) for 29 July - 19 August 2008. Table `motion` lists individual fish residence time intervals in each of three stream regions, while table `origin` lists the study-wide residence time of each fish and the stream region in which the fish was first tagged.

Format

Two data frames `motion` and `origin` with 1,140 rows and 149 rows of 4 variables, respectively.

Details

- from, to - start and end times as seconds since 1970-01-01 UTC (POSIXct)
- fish.id - unique identifier for each fish
- region - stream region (1 = 0 - 930 m, a cold downstream region with abundant and spawning sockeye salmon; 2 = 930 - 1360 m, a cold middle region with few if any sockeye salmon; 3 = > 1360 m, a warm upstream region where sockeye salmon were absent)

Source

Armstrong, J. B., D. E. Schindler, C. P. Ruff, G. T. Brooks, K. E. Bentley, and C. E. Torgersen. 2013. Diel horizontal migration in streams: juvenile fish exploit spatial heterogeneity in thermal and trophic resources. Ecology 94:2066-2075. DOI: doi: 10.1890/121200.1

is_events *Validate Event Table*

Description

Tests whether the object meets the basic requirements of an event table, i.e. a data frame containing at least two numeric, finite columns named 'from' and 'to' ordered such that `to` > or = `from` on all rows.

Usage

```
is_events(x, verbose = FALSE)
```

Arguments

`x`	An R object.
`verbose`	Logical value indicating whether to print the reason for test failure.

See Also

events, as_events, and read_events for creating valid event tables.

Examples

```
verbose <- TRUE
is_events(c(1, 3), verbose)
is_events(data.frame(from = 1, t = 3), verbose)
is_events(data.frame(from = 1, from = 1, to = 3), verbose)
is_events(data.frame(from = 1, to = TRUE), verbose)
is_events(data.frame(from = 1, to = NA), verbose)
is_events(data.frame(from = 3, to = 1), verbose)
is_events(data.frame(from = 1, to = 3), verbose)   # TRUE
```

---

netmap *Dungeness River (NetMap)*

Description

NetMap (terrainworks.com) output for the entire fluvial network of the Dungeness River (Washington, USA). NetMap employs digital elevation models to generate detailed river networks and compute biophysical variables for spatially continuous hydrologic units throughout the networks.

Format

A data frame with 16,616 rows and 47 variables.

Details

- CHAN_ID - channel identifier (1 = mainstem, all others are tributaries)
- OUT_DIST - distance upstream from the river mouth to the downstream end of the unit [km]
- LENGTH_M - unit length [m]
- ... - see NetMap's Master Attribute List

Source

https://terrainworks.com

plot_events *Plot Events as Bar Plots*

Description

Plots an event table as a grid of bar plots.

Usage

```
plot_events(
  e,
  group.col = NULL,
  groups = NULL,
  data.cols = NULL,
  dim = NULL,
  byrow = TRUE,
  main = NULL,
  xlabs = character(),
  ylabs = character(),
  xlim = NULL,
  ylim = NULL,
  xticks = NULL,
  yticks = NULL,
  xtick.labels = NULL,
  ytick.labels = NULL,
  plot.grid = FALSE,
  sigfigs = c(3, 3),
  col = NULL,
  border = par("fg"),
  lty = par("lty"),
  lwd = par("lwd"),
  xpd = FALSE,
  mar = c(2.1, 2.75, 1.5, 0.5),
  oma = c(2, 2, 2, 2),
  ...
)
```

Arguments

e	An event table.
group.col	Name or index of column defining the event grouping for plotting. If NULL, the events are treated as one group. Group NA is not plotted.
groups	Vector of values from group.col specifying which groups to plot. If NULL, all groups are plotted by order of first appearance in group.col.
data.cols	Names or indices of columns to plot, given as a list of character or numeric vectors. If multiple columns are specified, their bars are stacked together in one

plot. Names are interpreted as regular expressions (regex) matching full column names. If NULL, all columns not named from, to, or group.col are each plotted individually in order of appearance.

dim — The row and column dimensions of the grid. If NULL, the grid is column groups (rows) by event groups (columns) if byrow = TRUE, and event groups (rows) by column groups (columns) if byrow = FALSE.

byrow — Plots are added by column group, then bin group. If TRUE, plots are added by rows, rather than columns, to the grid.

main — Titles for each plot. If NULL, plots are titled by the column names, pasted together with separator " + ". Set main = NA to not title the plots.

xlabs, ylabs — Labels arranged at equal intervals along the bottom and left side of the plot grid. These are drawn in the outer margins of the figure, so oma[1] and oma[2] must be non-zero.

xlim, ylim — Limits for the x and y axes of all plots. If NULL, limits are set to the range of the data and the y limits extended as needed to include 0.

xticks, yticks — The positions of x and y tick marks for all plots. If NULL, only the min and max x and y are ticked (and 0 as needed for y). If axTicks, that function will be used to calculate R default tick mark positions. If NA, no ticks are drawn.

xtick.labels, ytick.labels — The labels for the x and y tick marks, coerced to character vectors and recycled as necessary. If NULL, the positions of the ticks are used as the labels, formatted with sigfigs. If NA, the tick marks are not labeled.

plot.grid — If TRUE, a lined horizontal grid is plotted at the yticks.

sigfigs — The maximum significant figures of the x and y axis labels.

col — Color(s) for the bars in each plot. If NA, bars are transparent. If NULL, a grey palette is used.

border — Color(s) for bar borders in each plot. If NA, borders are omitted.

lty — Line type(s) for bar borders in each plot.

lwd — Line width(s) for bar borders in each plot.

xpd — Logical value or NA. If FALSE, all plotting is clipped to the plot region, if TRUE, all plotting is clipped to the figure region, and if NA, all plotting is clipped to the device region.

mar — Numerical vector of the form c(bottom, left, top, right) giving the size of the inner margins of each plot in lines of text.

oma — Numeric vector of the form c(bottom, left, top, right) giving the size of the outer figure margins in lines of text.

... — Additional arguments passed to plot.

Details

Given a grouping variable for the rows of the event table (e.g., groups of bins of different sizes used in sample_events), and groups of columns to plot, bar plots are drawn in a grid for each combination of event and column groups. In each plot, the specified event table columns are plotted together as stacked bars. Negative and positive values are stacked separately from the y = 0 baseline.

Events with NA are not shown, differentiating them from zero-valued events which are drawn as thin black lines. Point events are drawn as thin vertical lines. Overlapping events are drawn as overlapping bars, so it is best to use sample_events with non-overlapping bins to flatten the data before plotting.

See Also

seq_events for generating groups of sequential bins, sample_events to populate groups of bins with event data.

Examples

```
e <- events(from = c(0, 10, 15, 25), to = c(10, 20, 25, 40), length = c(10, 10, 10, 15),
            x = c(1, 2, 1, 1), f = c('a', 'b', 'a', 'a'))
bins <- seq_events(event_coverage(e), c(8, 4, 2, 1))
e.bins <- sample_events(e, bins, list(sum, c('x', 'length')), scaled.cols = 'length')
plot_events(e.bins, group.col = 'group')
```

---

quinault *Quinault River Survey*

Description

An event table containing the results of a survey of the Quinault River (Washington, USA) in August 2009. Both physical variables and fish counts were collected.

Format

A data frame with 363 rows and 31 variables.

Details

- from, to - distance upstream from the river mouth [km]
- altitude - mean elevation above sea level [m]
- channel.type - channel type (1 = main, 2 = secondary)
- unit.type - unit type (P = pool, GP = glide-like pool, GR = glide-like riffle, R = riffle)
- unit.length - unit length [m]
- mean.width - mean wetted width [m]
- mean.depth - mean depth [m]
- max.depth - max depth [m]
- overhang.cover - channel banks with overhanging vegetation [%]
- boulder.cover - channel area covered by boulders [%]
- jams - number of log jams
- jam.area - total area of log jams [m^2]

- SACO.10/20/30/50/total - number of Bull Trout (*Salvelinus confluentus*) sized 10 - 20 cm / 20 - 30 cm / 30 - 50 cm / > 50 cm / total, respectively.
- ONXX.10/20/30/total - number of trout (*Oncorhynchus sp.*) sized 10 - 20 cm / 20 - 30 cm / > 30 cm / total, respectively.
- PRWI - number of Mountain Whitefish (*Prosopium williamsoni*)
- ONTS - number of Chinook Salmon (*Oncorhynchus tshawytscha*)
- ONMY - number of Rainbow Trout (*Oncorhynchus mykiss*)
- ONKI - number of Coho Salmon (*Oncorhynchus kisutch*)
- ONNE - number of Sockeye Salmon (*Oncorhynchus nerka*)
- ONGO - number of Pink Salmon (*Oncorhynchus gorbushcha*)
- ONKE - number of Chum Salmon (*Oncorhynchus keta*)
- CAMA - number of Largescale Sucker (*Catostomus macrocheilus*)
- LATR - number of Pacific Lamprey (*Lampetra tridentata*)

Source

Samuel J. Brenkman (National Park Service, Olympic National Park, Washington, USA), unpublished data.

`read_events` *Read File as Event Table*

Description

Reads a file in table format and attempts to coerce it to an event table.

Usage

```
read_events(file, from.col = 1, to.col = 2, sep = "", header = TRUE, ...)
```

Arguments

`file`	Name, `connection`, or `url` of the file to be read as an event table.
`from.col, to.col`	Names or indices of the columns containing event endpoints. Values are swapped as needed to ensure that `to` > `or` = `from` on all rows.
`sep`	Character separating values on each line of the file. If `sep = ""` (the default), the separator is 'white space' (that is, any combination of one or more spaces, tabs, newlines and carriage returns).
`header`	Logical value indicating whether the file contains column names as its first line. If `FALSE`, columns will be named "V" followed by the column number, unless `col.names` (a vector of optional column names) is provided as an additional argument.
`...`	Additional arguments, of the form `tag = value`, to be passed directly to `read.table` to control how the file is read.

Details

The file is read into R by calling `read.table`. Any of its arguments can be set by passing additional `tag = value` pairs. `from.col` and `to.col` are renamed to "from" and "to" as needed. Since these column names must be unique, other columns cannot also be called "from" or "to".

See Also

`read.table`.

`events` and `as_events` for creating event tables from existing objects.

---

sample_events *Sample Events*

Description

Computes event table variables over the specified sampling intervals, or "bins".

Usage

```
sample_events(
  e,
  bins,
  ...,
  scaled.cols = NULL,
  col.names = NULL,
  drop.empty = FALSE
)
```

Arguments

`e`	An event table.
`bins`	An event table specifying the intervals for sampling.
`...`	Lists specifying the sampling functions and parameters to be used (see the `Details`).
`scaled.cols`	Names or indices of the event columns to be rescaled after cutting (see `cut_events`). Names are interpreted as regular expressions (`regex`) matching full column names.
`col.names`	Character vector of names for the columns output by the sampling functions. If `NULL`, the columns are named automatically (see the `Details`).
`drop.empty`	If `TRUE`, bins not intersecting any events are dropped.

Details

Events are cut at bin endpoints, and any `scaled.cols` columns are rescaled to the length of the resulting event segments. The event segments falling into each bin are passed to the sampling functions to compute the variables for each bin. Bins sample from events they overlap: line events with whom they share more than an endpoint, or point events with equal endpoints (if the bin itself is a point).

Sampling functions are specified in lists with the format `list(FUN, data.cols, by = group.cols, ...)`. The first element in the list is the function to use. It must compute a single value from one or more vectors of the same length. The following unnamed element is a vector specifying the event column names or indices to recursively pass as the first argument of the function. Names are interpreted as regular expressions (`regex`) matching full column names. Additional unnamed elements are vectors specifying additional event columns to pass as the second, third, ... argument of the function. The first "by" element is a vector of event column names or indices used as grouping variables. Any additional named arguments are passed directly to the function. For example:

list(sum, 1:2, na.rm = TRUE) => sum(events[1], na.rm = TRUE), sum(events[2], na.rm = TRUE) list(sum, 1, 3:4, 5) => sum(events[1], events[3], events[4], events[5]), ... list(sum, c('x', 'y'), by = 3:4) => list(sum, 'x'), list(sum, 'y') grouped into all combinations of columns 3 and 4

Using the latter example above, column names are taken from the first argument (e.g. `x, y`), and all grouping variables are appended (e.g. `x.a, y.a, x.b, y.b`), where `a` and `b` are the levels of columns 3 and 4. `NA` is also treated as a factor level. Columns are added left to right in order of the sampling function arguments. Finally, names are made unique by appending sequence numbers to duplicates (using `make.unique`).

Value

The `bins` event table with the columns output by the sampling functions appended.

See Also

`seq_events` to generate sequential bins.

Examples

```
e <- events(from = c(0, 10, 15, 25), to = c(10, 20, 25, 40), length = c(10, 10, 10, 15),
            x = c(1, 2, 1, 1), f = c('a', 'b', 'a', 'a'))
bins <- rbind(seq_events(event_coverage(e), 4), c(18, 18))
sample_events(e, bins, list(sum, 'length'))
sample_events(e, bins, list(sum, 'length'), scaled.cols = 'length')
sample_events(e, bins, list(sum, 'length', by = 'f'), scaled.cols = 'length')
sample_events(e, bins, list(weighted.mean, 'x', 'length'), scaled.cols = 'length')
sample_events(e, bins, list(paste0, 'f', collapse = "."))
```

seq_events *Generate Sequential Events*

Description

Generates groups of regularly sequenced events fitted to the specified intervals. Intended for use as bins with sample_events.

Usage

```
seq_events(coverage, length.out = NULL, by = NULL, adaptive = FALSE)
```

Arguments

coverage	An event table specifying the non-overlapping intervals to which the event sequences will be fitted. Gaps in coverage do not count towards event length. Points in the coverage are currently ignored.
length.out	The number of events in each sequence. Event lengths are chosen such that they evenly divide the coverage.
by	The length of the events in each sequence. Ignored if length.out is defined. When the length does not evenly divide the coverage, a shorter event is appended to the end of the sequence.
adaptive	If TRUE, events are adjusted locally so that a whole number of events fit within each coverage interval, preserving breaks and gaps.

Value

An endpoint-only event table with an additional group field if the length of length.out or by is > 1.

See Also

event_range, event_coverage, and fill_event_gaps for building a coverage from an existing event table.

Examples

```
e <- events(c(0, 20, 40), c(10, 30, 45))
no.gaps <- event_range(e)
has.gaps <- event_coverage(e)
seq_events(no.gaps, by = 10)                          # unequal length (last is shorter)
seq_events(no.gaps, by = 10, adaptive = TRUE)          # equal length (11.25)
seq_events(no.gaps, length.out = 4)                    # equal length (11.25)
seq_events(has.gaps, length.out = 4, adaptive = FALSE) # equal coverage (11.25), straddling gaps
seq_events(has.gaps, length.out = 4, adaptive = TRUE)  # unequal coverage, fitted to gaps
seq_events(no.gaps, length.out = c(2, 4))              # "group" column added
```

simple *Simple Event Table*

Description

A simple, hypothetical event table.

Format

A data frame with 11 rows and 7 variables.

Details

- from, to - endpoint positions
- x, y, z - numeric variables
- factor - a factor variable

sort_events *Sorted Events*

Description

`sort_events` sorts events by ascending `from`, then ascending `to`. `is_unsorted_events` tests whether the events are not sorted, without the cost of sorting them.

Usage

```
sort_events(e)

is_unsorted_events(e)
```

Arguments

`e`	An event table.

Examples

```
e <- events(c(1, 1, 3, 2), c(2, 1, 4, 3))
is_unsorted_events(e)
sort_events(e)
```

to_date *Convert event endpoints to dates*

Description

Convert event endpoints to dates

Usage

```
to_date(e, origin = as.Date("1970-01-01"))
```

Arguments

e	Event table or atomic vector.
origin	Date object (see as.Date).

Examples

```
t <- as.Date("1970-01-01") + 0:4
e <- events(t)
to_date(e)
to_date(e$from)
```

to_datetime *Convert event endpoints to date-times*

Description

Convert event endpoints to date-times

Usage

```
to_datetime(e, tz = "UTC", origin = as.POSIXct("1970-01-01", tz = "UTC"))
```

Arguments

e	Event table or atomic vector.
tz	Time zone (see timezones).
origin	Date-time object (see as.POSIXct).

Examples

```
t <- as.POSIXct("1970-01-01", tz = "UTC") + 0:4
e <- events(t)
to_datetime(e)
to_datetime(e$from)
```

transform_events *Transform Events*

Description

Transforms events by scaling, then translating their endpoint positions. That is, the transformed `[from, to] = scale * [from, to] + translate`.

Usage

```
transform_events(e, scale = 1, translate = 0)
```

Arguments

`e`	An event table.
`scale`	Number by which event endpoints should be scaled.
`translate`	Number by which event endpoints should be translated.

Examples

```
e <- events(c(10, 100), c(100, 1000))
transform_events(e, scale = 2, translate = 1)
```

Index